

Scalasca Parallel Performance Analyses of PEPC

Zoltán Szebenyi^{1,2}, Brian J. N. Wylie¹, and Felix Wolf^{1,2}

¹ Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany

² Aachen Institute for Advanced Study in Computational Engineering Science,
RWTH Aachen University, Germany
{z.szebenyi,b.wylie,f.wolf}@fz-juelich.de

Abstract. PEPC (Pretty Efficient Parallel Coulomb-solver) is a complex HPC application developed at the Jülich Supercomputing Centre, scaling to thousands of processors. This is a case study of challenges faced when applying the Scalasca parallel performance analysis toolset to this intricate example at relatively high processor counts. The Scalasca version used in this study has been extended to distinguish iteration/timestep phases to provide a better view of the underlying mechanisms of the application execution. The added value of the additional analyses and presentations is then assessed to determine requirements for possible future integration within Scalasca.

Keywords: Parallel/distributed systems, performance measurement & analysis tools, application tracing & profiling.

1 Introduction

PEPC [7] is a 3-dimensional particle simulation code which employs a hierarchical, parallel tree algorithm implemented using MPI to compute the forces on the particles. The code is presently used for various applications in plasma physics and astrophysics. According to the author of the code, potential bottlenecks lie in the domain decomposition routine, tree construction and tree ‘walk’, the last of which requires significant point-to-point communication of multipole information between processors, and is thereby sensitive to load imbalance.

This work presents Scalasca measurements and analyses of the application on the IBM BlueGene/P system of the Jülich Supercomputing Centre. The Scalasca toolset [1,2] is a highly scalable performance analysis toolset capable of both taking runtime summaries or collecting and automatically analyzing event traces. The latter can be searched for complex event patterns which may indicate important performance bottlenecks. Scalasca performs trace analysis in parallel on the same number of processors that was used to originally take the measurement.

In this work PEPC is analyzed using an extended version of Scalasca [6], providing the additional capability of phase instrumentation. This means that after manually identifying the main time-stepping loop of the application and inserting markers around the loop body, subsequent Scalasca measurements become aware of individual iterations and support the analysis of the time-dependent

behavior or of individual iterations, which corresponds to dynamic phases in TAU [4]. After considering examples of such PEPC execution analyses and presentations, these are assessed to determine requirements for the potential future integration of the corresponding capabilities with Scalasca.

2 Experimental Results

A 1024-way test case, which was run for 1,300 timesteps on a BlueGene/P system, is used here as an example to show what we have learned about the performance characteristics of the application using the tools provided by the extended Scalasca toolset. This example was chosen as it shows not only interesting patterns of time-varying behavior, but also performance problems, like serious load imbalance growing rapidly over time.

Figures 1&2 show an example of the usefulness of having analysis data individually for all the iterations instead of having just a summary of the whole program execution. Figure 1 shows the case where the iterations are not distinguished and only aggregate metrics are available. When looking at the point-to-point communication metric, a few processes appear as hot-spots in the topology pane, showing that there is some imbalance. This is important to recognize, however, additional insight can be extracted from the extended, phase-instrumented analysis. When selecting in turn the individual iterations distinguished in Fig. 2, the execution behavior can be observed evolving over time. During the first iterations, the communication is relatively balanced and there are no extreme hot-spots visible. Over time some hot-spots appear and become increasingly pronounced, and they move at different rates from one MPI rank to the next. The number of hot-spots diminishes towards the end of the 1,300 steps, however, the severities of the highest ones are increasing rapidly.

Figure 3 shows four different views of the same metric, *point-to-point communication count* (i.e., the number of sends and receives), to compare the different kinds of information they provide. The *Phase graph* (upper left) gives an overview of the evolution of the values over time. The meaning of the different colors is as follows: for each timestep, the value for the process with the largest time is shown in red, the median is shown in blue, and the smallest is shown in green. Looking at this graph it is obvious that there is a serious communication imbalance in the application, as the minimum and median values are relatively low and constant throughout the execution, but the maximum value is growing rapidly, and it rises many times higher than the minimum or median values. This means that within each timestep most of the processes are behaving in a relatively balanced fashion, but that there are a few processes being involved in many more communications.

Which processes are responsible for the imbalance? On the *Process graph* (upper right), the *x*-axis shows the different process ranks, so the different colors now distinguish the minimum, median and maximum value during the different timesteps for the given process rank. It is not obvious which process is responsible as many processes show very high values, but again these can only be for a few

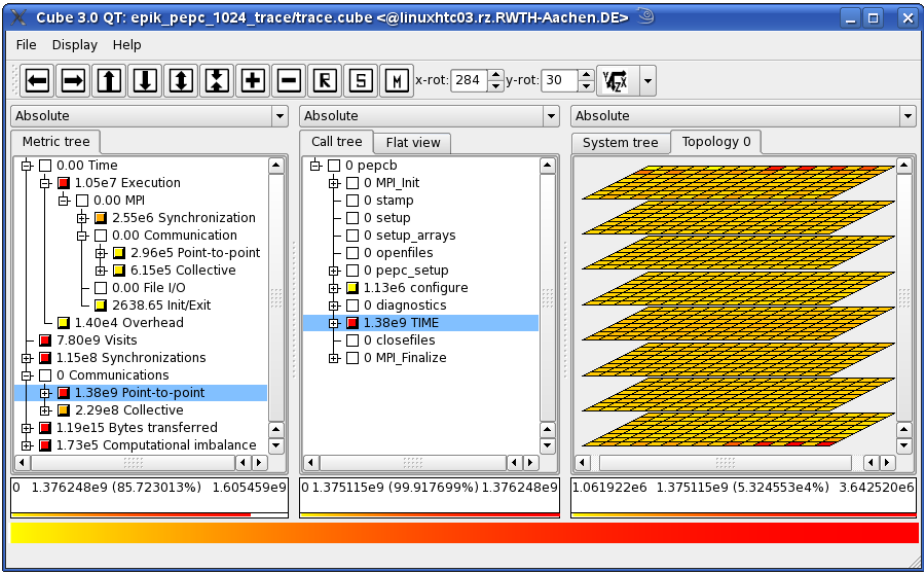


Fig. 1. Scalasca analysis report explorer showing a PEPC runtime summary with significant imbalance in the number of point-to-point communications (selected in left pane). The 1,024 application processes are arranged according to the BlueGene/P physical network topology (right pane).

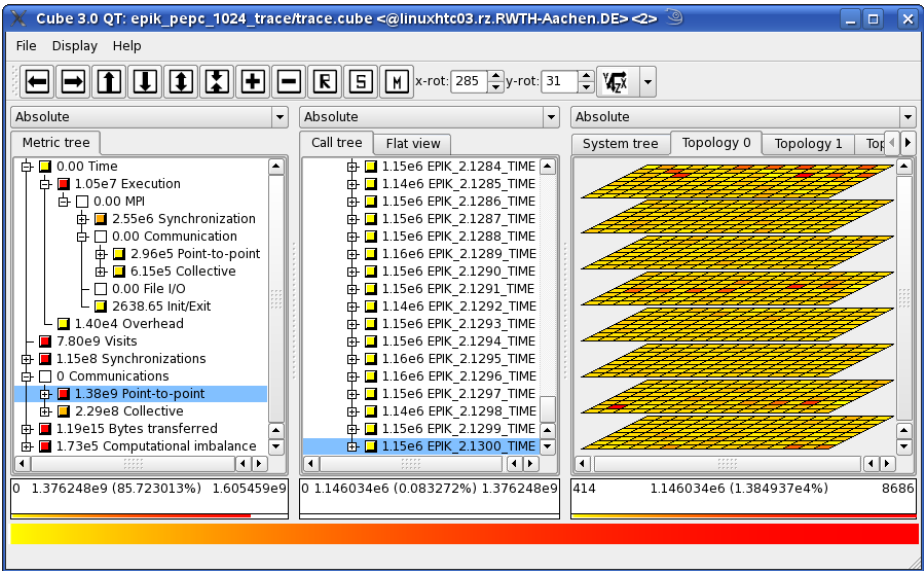


Fig. 2. Scalasca analysis report explorer showing a PEPC runtime summary with timesteps distinguished via phase instrumentation (centre pane)

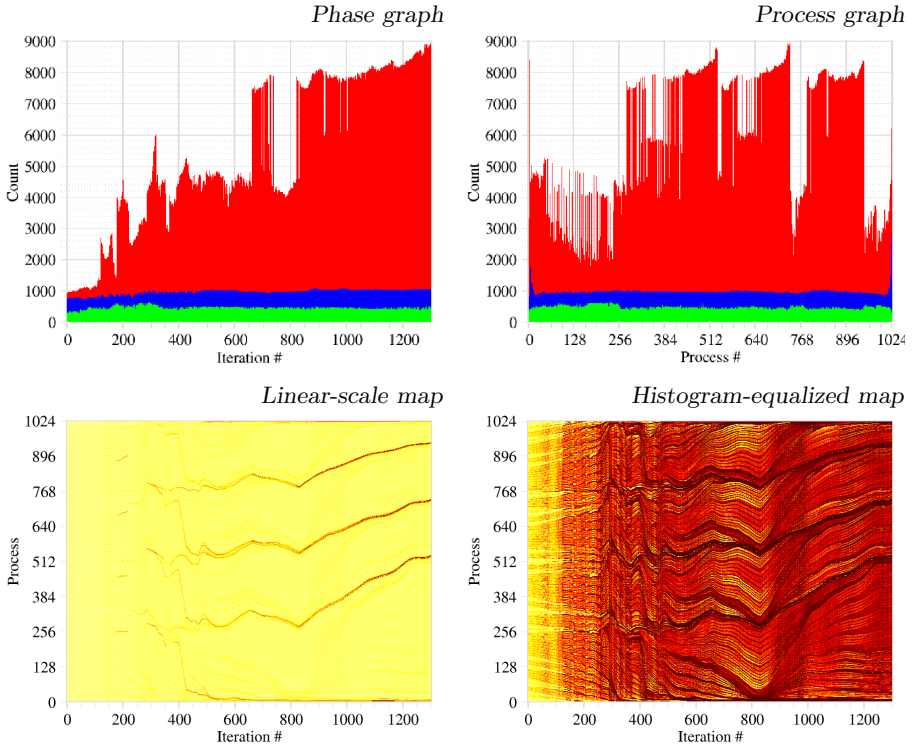


Fig. 3. Different analysis presentations of point-to-point communication count metric

timesteps, as the minimum and median values are low for all of them (except for the first and last few ranks where the median is somewhat higher).

The case is getting increasingly confusing, and we still do not see what is going on here with these high values, but the *Linear-scale map* on the lower left makes it much clearer. On a linear-scale map, the x -axis shows the iteration number, the y -axis the process rank, and the values are color-coded from light yellow (for the lowest value) to dark red (for the highest value, here 9000). The map shows that at the beginning all processes start off relatively balanced, however, after a few hundred timesteps a low number of hot-spots gradually appear whose values get much higher over time than the average. What is interesting about these hot-spots is that they are not bound to any given process, but rather they move to neighboring processes in a systematic and coordinated manner. As they migrate some hot-spots appear to merge, so after around 700 timesteps only five hot-spots remain, each consisting of a few processes.

This movement of the hot-spots is responsible for the confusing values seen in the *Phase* and *Process* graphs, so understanding their behavior in more detail is useful. Taking a closer look at the light yellow area of the *Linear-scale map* reveals that the values in the background are not exactly the same. There are also some patterns there, but they are not very easy to see as their differences

are small compared to the range of the graph. On the *Histogram-equalized map*, light yellow and dark red still mean the same lowest and highest values as on the *Linear-scale map*, but here the histogram of the map values is equalized so that every color level is used for approximately the same number of points. This produces the maximum contrast on the map and reveals previously invisible details: there are many more systematic details down to the finest granularity than there are visible on the *Linear-scale map*.

Figure 4 shows more metric graphs of the test case execution, including the time taken by different activities and the number of bytes transferred in each timestep. Where no explicit legend is given, the different colors have the same meaning as before.

The *Execution Time Breakdown* graph uses a different coloring where yellow is the average time each process spent in pure computation (i.e. non-MPI functions), the small green part is the useful time spent in MPI communication, orange is the blocking time in situations like late sender, and magenta is synchronization time at barriers. These values combine to the total execution time of each timestep, averaged over all the processes. Execution time does not show any differences between processes as execution times are synchronized every timestep due to the collective communications.

The high peak values every 100th iteration are due to checkpointing. Also notable on this graph is the evolution of the total execution time per iteration along the 1,300 timesteps, gradually increasing more than twofold from around 5.5 seconds to more than 12 seconds. From the breakdown this is seen to be due to the computational workload itself growing over time (as the pure computation time is growing in the same way), however, MPI blocking and synchronization times are growing as well.

On the *Execution Time Proportions* graph, the same data is normalized for each iteration to show the fraction of execution time spent in each activity. This graph shows that the proportion of pure computation time is shrinking from around 78% down to 73%, while the proportion of MPI blocking time grows from 3% to 6% and the proportion of MPI synchronization grows from 18% to 19%. MPI blocking time is therefore the fastest growing problem, even though the time spent in synchronizations is still higher.

The *Point-to-point Communication Time* graph shows the load imbalance very clearly, as the median and maximum times spent in point-to-point communications are much higher than the minimum, and the median is around halfway between the minimum and maximum. This means that the imbalance in point-to-point communication time involves many or most of the processes. Comparing with *Point-to-point Late Sender Time*, most of the time is actually spent in situations where the receiver blocked waiting for the corresponding send to be initiated, which suggests that some point-to-point messages are not sent in time. This causes a majority of processes to spend more time in point-to-point communication than is absolutely necessary (which is around the minimum value shown in green).

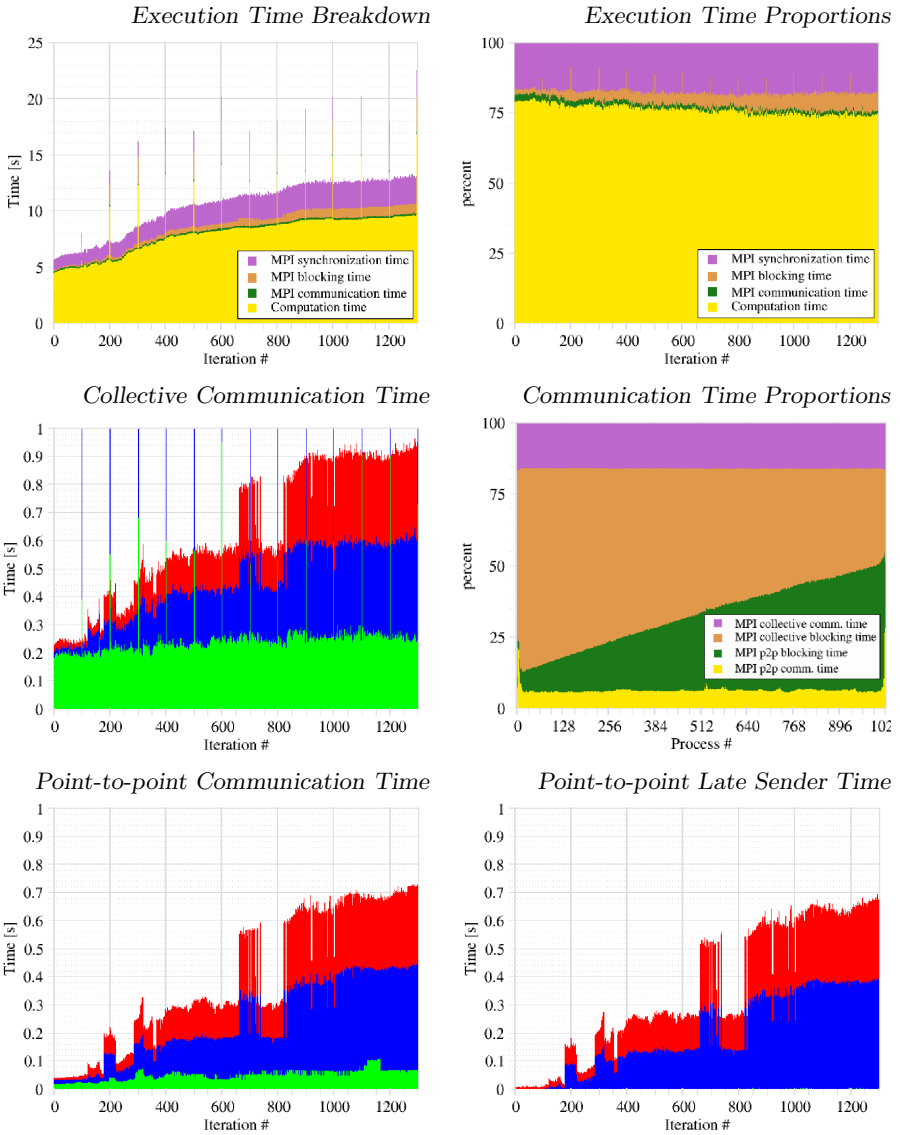


Fig. 4. Graphs of time and bytes transferred in different communications in PEPC

The *Collective Communication Time* graph also shows an imbalance very much like that seen on its point-to-point counterpart, however, with a much higher minimum. Apparently, the minimum time for collective communication in each timestep is longer than the corresponding point-to-point time, but everything in excess of this minimum closely resembles the point-to-point late sender time. The high peaks every 100th iteration are due to checkpointing activity in those timesteps.

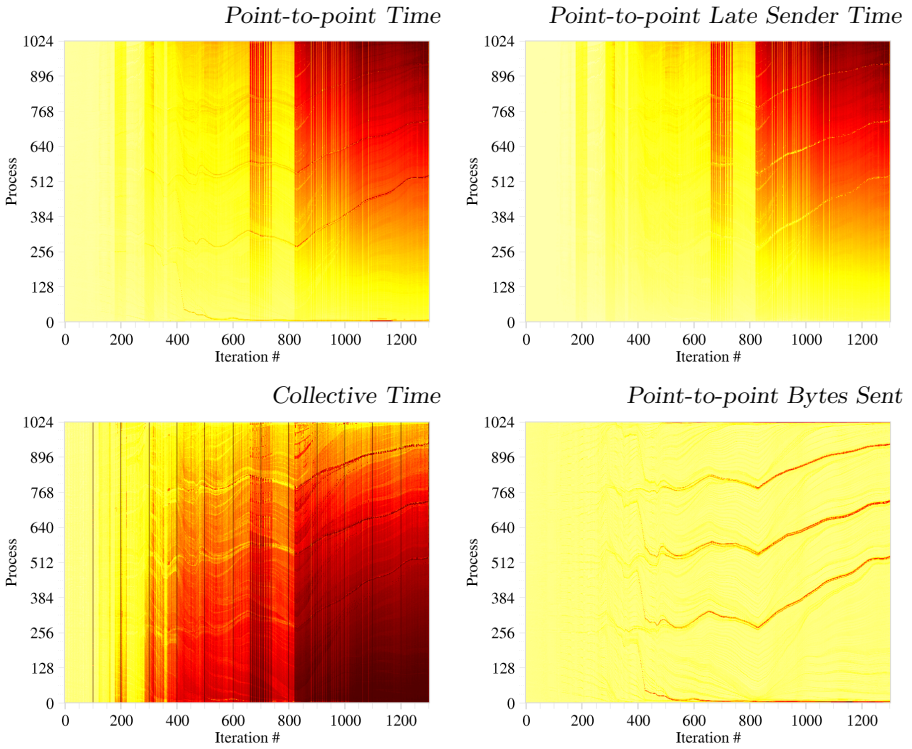


Fig. 5. Maps of time and bytes transferred in different communications in PEPC

Figure 5 shows some maps that help clarify the nature of the communication time imbalance. The *Point-to-point Time* map shows how the above mentioned communication time imbalance is distributed among the processes in a very specific and systematic way. Generally the higher the MPI process rank, the more time it spends in point-to-point communication. There are some exceptions to this rule, as the processes which send more point-to-point messages take somewhat longer than their neighbors, particularly the hot-spots which all have very high communication times.

The *Point-to-point Late Sender Time* map shows a rather similar distribution, with one major difference being the hot-spots. These show very low waiting time in late sender situations, but they have a very high amount of time spent in point-to-point communication. This means that they really communicate a lot and then do not spend much time waiting, while all the others that must wait for them do.

The *Collective Time* map also shows something interesting. It seems to be the inverse of Point-to-point time, in the sense that the order of MPI ranks is reversed here. The higher the MPI rank, the less time it spends in collective communications. This suggests that the price of the communication imbalance in point-to-point communications must be paid again in collective communications. First point-to-point

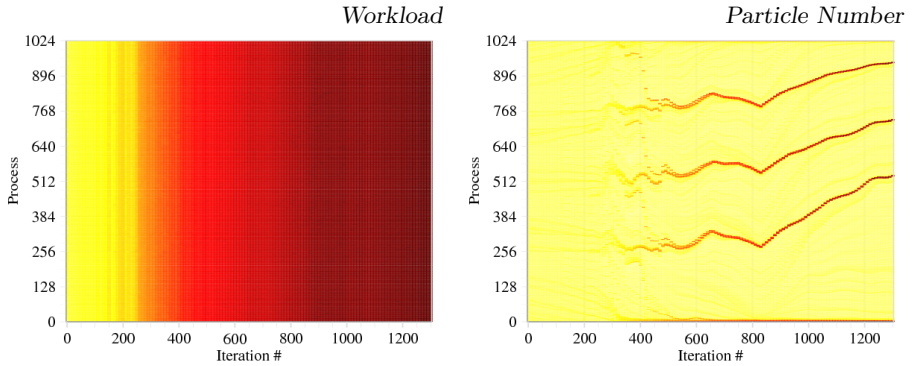


Fig. 6. Maps of application-specific metrics from log files written every 10th timestep

communication causes the processes to get out of balance, then the next collective communication synchronizes them again, and the processes which wait less in point-to-point and are slightly ahead of the others have to wait more while the ones that are later wait less. So in the end they accumulate the same amount of blocking time, which, however, is distributed in reversed fashion across the processes.

This phenomenon is clearly shown in the *Communication Time Proportions* graph in Fig. 4, which shows the process ranks on the x -axis, and the proportion of MPI point-to-point communication time (yellow), point-to-point blocking time (green), collective blocking time (orange) and collective communication time (magenta) on the y -axis. The most important aspect of this graph is the diagonal border between point-to-point blocking time and collective blocking time. The higher the MPI rank, the more point-to-point blocking time and the less collective blocking time was diagnosed, with both waiting time categories consuming around 80% of the total communication time. This highlights how seriously this problem degrades communication efficiency.

But what causes the imbalance? Examining the *Point-to-point Bytes Sent* graph in Fig. 5 it is clear that the hot-spot processes are sending much more data than any other process. *Point-to-point Bytes Sent* and *Received* graphs (not shown due to space restrictions) reveal that while all processes receive around the same amount of data, the amount varies considerably between senders. Therefore the hot-spot processes send much more data than other processes, and they send data to all. This suggests that the hot-spot processes are the communication bottleneck, as they send much more data than the others. Moreover, the data are sent in process rank order, such that higher-ranked processes have to wait for them to complete sending data to lower ranked processes before they receive any data themselves.

Figure 6 shows maps of two application-specific metrics extracted from application log files written every 10th timestep. On the *Workload* map we see the workload metric calculated by the application. According to the developers, PEPC runs a workload-balancing algorithm every timestep where it balances this metric, therefore it is no surprise that it really is balanced over the processes. It further

shows some growth over time which correlates to the growing computational part of the execution time. On the *Particle Number* map we see the number of simulated particles assigned to each process. This means that the workload-balancing algorithm assigns a very large number of particles to the hot-spot processes which in turn causes a communication bottleneck to appear on those processes and leads to the communication imbalance.

The TAU Paraprof 3D visualizer [4] is a third-party tool able to visualize Scalasca analysis reports. 3D visualization is a promising complement to 2D maps, as it can be easier to compare scales of data at different positions by comparing the height of bars as opposed to comparing the brightness of colors.

3 Conclusion

In our analysis of the PEPC execution on BlueGene/P we have identified some complex performance patterns, and have found a good potential for communication performance improvement. The PEPC developer team is actively working on finding the causes that led to the serious imbalance in particle numbers that we identified as the root cause of the point-to-point communication problem. They are also looking at ways to modify the workload-balancing algorithm to avoid this situation.

The depth of analysis we conducted in this case would not have been possible without our prototype Scalasca extensions, namely phase instrumentation and the different visualization techniques used to make the huge amounts of data collected accessible to the user. During the course of the analysis, we have found all the different kinds of visualization techniques (phase and process graphs, linearly-scaled and histogram-equalized 2D maps, 3D visualization) useful in many ways, as the insights they provide often complement each other.

We have also found that there are serious limitations concerning the visualization of the huge amounts of data collected, as it can easily happen that on the monitor or printer used for displaying the data, it is not possible to get sufficient resolution to have one pixel for each process or iteration. Other groups also identified this issue [8,9] and found different solutions, like zooming in on the visualized data while also displaying a miniature map of the whole chart marking the zoomed region, or letting the user choose from different display options when there is more than one data point for a single pixel, such as the maximum, minimum, sum, or median of the values. These techniques can provide partial solutions to the problem, but further investigation of this topic could also prove to be valuable in the future.

Visualizing time-dependent behavior with an animation, where the user can step through iterations to track changes of a metric over time, also seems to be an interesting possibility that proved useful in understanding PEPC measurement results. Furthermore, 3D visualization such as the one offered by TAU was found to be extremely valuable and perhaps the most insightful of all the different visualization techniques, but also the technically most challenging.

References

1. Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Frings, W., Furlinger, K., Geimer, M., Hermanns, M.-A., Mohr, B., Moore, S., Pfeifer, M., Szebenyi, Z.: Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications, in *Tools for High Performance Computing*. In: Proc. 2nd Int'l Workshop on Tools for High Performance Computing, Stuttgart, Germany, pp. 167–181. Springer, Heidelberg (2008)
2. Jülich Supercomputing Centre, Scalasca toolset for scalable performance analysis of large-scale parallel applications, <http://www.scalasca.org/>
3. Wylie, B.J.N., Wolf, F., Mohr, B., Geimer, M.: Integrated runtime measurement summarization and selective event tracing for scalable parallel execution performance diagnosis. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 460–469. Springer, Heidelberg (2007)
4. Malony, A.D., Shende, S.S., Morris, A.: Phase-based parallel performance profiling, In: *Parallel Computing: Architectures, Algorithms and Applications*. Proc. 11th ParCo Conf., Málaga, Spain, October 2006. NIC Series, vol. 33, pp. 203–210. John von Neumann Institute for Computing, Jülich (2006)
5. Furlinger, K., Gerndt, M., Dongarra, J.: On using incremental profiling for the performance analysis of shared-memory parallel applications. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 62–71. Springer, Heidelberg (2007)
6. Szebenyi, Z., Wylie, B.J.N., Wolf, F.: SCALASCA parallel performance analyses of SPEC MPI2007 applications. In: Kounev, S., Gorton, I., Sachs, K. (eds.) SIPEW 2008. LNCS, vol. 5119, pp. 99–123. Springer, Heidelberg (2008)
7. Gibbon, P., Frings, W., Dominiczak, S., Mohr, B.: Performance Analysis and Visualization of the N-Body Tree Code PEPC on Massively Parallel Computers. In: Proc. 11th ParCo Conf., Málaga, Spain, October 2006. NIC Series, vol. 33, pp. 367–374. John von Neumann Institute for Computing, Jülich (2006)
8. Labarta, J., Giménez, J., Martínez, E., González, P., Servat, H., Llort, G., Aguilar, X.: Scalability of Visualization and Tracing Tools. In: Proc. 11th ParCo Conf., Málaga, Spain, October 2006. NIC Series, vol. 33, pp. 869–876. John von Neumann Institute for Computing, Jülich (2006)
9. Brunst, H., Nagel, W.E.: Scalable Performance Analysis of Parallel Systems: Concepts and Experiences. In: Proc. 10th ParCo Conf., Dresden, Germany, September 2003, pp. 737–744 (2003)